Relevance Heuristics for Program Analysis

K. L. McMillan
Cadence Research Labs
mcmillan@cadence.com

Abstract

Relevance heuristics allow us to tailor a program analysis to a particular property to be verified. This in turn makes it possible to improve the precision of the analysis where needed, while maintaining scalability. In this talk I will discuss the principles by which SAT solvers and other decision procedures decide what information is relevant to a given proof. Then we will see how these ideas can be exploited in program verification using the method of Craig interpolation. The result is an analysis that is finely tuned to prove a given property of a program. At the end of the talk, I will cover some recent research in this area, including the use of interpolants for verifying heap-manipulating programs.

Categories and Subject Descriptors F.3.1 [*Specifying and Verifying and Reasoning about Programs*]: Mechanical verification

General Terms Languages, Theory, Verification

Keywords abstract interpretation, model checking, Craig interpolation

Summary

Static analysis of programs using abstract interpretation methods has proved an effective technique for establishing key properties of programs for compiler optimization or verification purposes. There is, however, an inherent tension in these methods between precision and cost of the analysis. An abstract domain that maintains sufficient information about the program state to verify a given property may be too costly to allow scaling to programs of even moderate size.

Automated abstraction refinement methods address this problem by tailoring the abstraction to the property. This is usually done in an iterative manner, by analyzing abstract counterexamples, or failures of the abstraction to prove the given property for a particular program execution trace. After each such failure, information is added to the abstract domain that is sufficient to prevent the failure in subsequent analysis. This approach is known as CEGAR, or counterexample-guided abstraction refinement.

Such methods have shown significant promise in analyzing low-level software, such as operating system device drivers, and other control-oriented codes. For example, the SLAM toolkit from Microsoft research has proved effective in finding control errors (such as illegal use of kernel API functions) in real-world device driver codes. SLAM is based on predicate abstraction, a parameterized abstract interpretation that effectively computes the strongest inductive invariant of a program that can be expressed as a Boolean combination of a given set of atomic predicates. Choosing the predicates by counterexample-guided refinement gives SLAM the ability to focus the abstraction on information that is relevant to the proof (or falsification) of a given property. This ability allows SLAM and similar tools to scale to real codes of moderate size, albeit only in the case when the property is fairly shallow, in the sense that it requires only a small amount of information about the program's state to prove it.

This talk will focus on the key question in all such methods: how do we know what information about the program is *relevant* to proving a given property? Though this might seem like an ill-defined question, in fact there has been extensive research into this question in the area of Boolean satisfiability (SAT) solvers. This has resulted in algorithms for SAT that are quite effective in focusing proofs on relevant facts and ignoring extraneous facts. This ability has been exploited in other areas, such as finite-state verification and ground decision procedures (using an approach known as SMT, or SAT modulo theories).

The relevance heuristics in SAT solvers are based on a simple idea: that facts useful in proving special cases are likely to be useful in general, if we can figure out how to generalize them appropriately. This same idea is implicit in CEGAR. That is, in proving a property of a specific program execution, we hope to discover predicates that are useful for proving the property for all executions. For program analysis, a useful "special case" is typically some restricted execution of the program, which might be obtained, for example, by unwinding the program loops a finite number of times.

The first part of the talk will cover relevance heuristics based on the method of Craig interpolation. In effect, this method allows us to construct a Floyd/Hoare proof for a loop-free program fragment based on a proof generated by an off-the-shelf decision procedure or theorem prover. This in turn allows us to exploit the prover's built-in relevance heuristics to refine our abstract domains. We will also observe that relevance heuristics give us a way to prove properties using abstract domains of infinite height, without the risk of over-widening.

The second part of the talk will focus on recent developments in relevance heuristics for heap-manipulating programs. We will see how the Craig interpolation approach can be extended to handle the richer logical framework needed to express properties of heaps. In particular, we must be able to handle quantifiers (to reason about heaps of unbounded size) and reachability of linked structures (to express properties such as acyclicity). The talk will outline one potential solution to these problems, and show how interpolation in first-order logic with transitive closure might be used for abstraction refinement in verifying heap-manipulating programs. The ability to focus such abstraction on facts relevant to a given property

POPL'08, January 7–12, 2008, San Francisco, California, USA.
ACM 978-1-59593-689-9/08/0001.

may in turn allow the methods to scale up to programs of practical size.

About the speaker

Ken McMillan is currently a research scientist at Cadence Berkeley Labs in Berkeley, California. He works in formal verification, primarily in model checking and compositional methods. He holds a BS in electrical engineering from the University of Illinois at Urbana (1984), an MS in electrical engineering from Stanford (1986) and a Ph.D. in computer science from Carnegie Mellon (1992). He is the author of the SMV symbolic model checker, and received the 1992 ACM doctoral dissertation award for his thesis on symbolic model checking. For his work in this area, he also received an SRC technical excellence award (1995), the ACM Paris Kannelakis award (1998), and the Alan Newell award from Carnegie Mellon (1998). His current research is focused on using Craig interpolation methods for software verification.